Dataflow Analysis for Multiprocessor Systems with Non-Starvation-Free Schedulers

Joost P.H.M. Hausmans §
joost.hausmans@utwente.nl

Stefan J. Geuns §
stefan.geuns@utwente.nl

Maarten H. Wiggers
maarten.wiggers@gmail.com

Marco J.G. Bekooij ¶ §
marco.bekooij@nxp.com

§ University of Twente, Enschede, The Netherlands

¶ NXP Semiconductors, Eindhoven, The Netherlands

ABSTRACT

Dataflow analysis techniques are suitable for the temporal analysis of real-time stream processing applications. However, the applicability of these models is currently limited to systems with starvation-free schedulers, such as Time-Division Multiplexing (TDM) schedulers. Removal of this limitation would broaden the application domain of dataflow analysis techniques significantly.

In this paper we present a temporal analysis technique for Homogeneous Synchronous Dataflow (HSDF) graphs, that is also applicable for systems with non-starvation-free schedulers. Unlike existing dataflow analysis techniques, the proposed analysis technique makes use of an enabling-jitter characterization and iterative fixed-point computation.

The presented approach is applicable for arbitrary (cyclic) graph topologies. Buffer capacity constraints are taken into account during the analysis and sufficient buffer capacities can be determined afterwards. The approach presented in this paper is the first approach that considers non-starvation-free schedulers in combination with arbitrary HSDF graphs

The proposed dataflow analysis technique is implemented in a tool. This tool is used to evaluate the analysis technique using examples that illustrate some important differences with other temporal analysis methods. The case-study discusses how the method presented in this paper can be used to solve a problem with the inaccuracy of the temporal analysis results of a real-time stream processing system. This stream processing system consists of an FM receiver together with a DAB receiver application which both share a Digital Signal Processor (DSP).

1. INTRODUCTION

Real-time stream processing applications are often executed on multiprocessor systems and require analysis methods to guarantee their temporal constraints. Dataflow models can often be used to intuitively model the temporal behavior of such applications [9, 15] and several analysis algorithms exist to verify the temporal constraints [4]. Algorithms have also been developed for the computation of buffer sizes [12, 18] and the temporal analysis of systems with run-time schedulers [19, 20, 10]. However, the scope of these analysis methods is limited to the class of starvation-free scheduling algorithms [20]. For such starvation-free schedulers, the interference from other tasks can be bounded by only knowing the execution time of those tasks. For the broader class of non-starvation-free schedulers, the interference from other tasks can only be bounded by knowing how often the tasks are started. Support for non-starvation-free schedulers would increase the scope of dataflow analysis techniques significantly.

Other methods exist which support the temporal analysis of systems with non-starvation-free schedulers. These methods use traffic propagation to give guarantees on the temporal behavior of applications [5, 2]. Due to this traffic propagation, the correlation between dependent data flows is lost [7] with the result that latency constraints cannot be taken into account accurately during the analysis. They can only be checked afterwards [14]. Furthermore, not all graph topologies are supported [13]. Methods exist which consider arbitrary cyclic data dependencies [16]. Also cyclic resource dependencies caused by non-starvation-free schedulers can be handled [6] but the combination of cyclic data and resource dependencies is not yet considered.

In this paper we propose a temporal analysis flow based on dataflow analysis that can be used to analyze systems with non-starvation-free schedulers. Thanks to the properties of the dataflow graph abstraction, cyclic constraints in an application can be taken into account during the analysis. The same holds for buffer size constraints. Furthermore, the proposed analysis method has the property that bursts only have to be taken into account once, for the latency of a path. This phenomenon is known as the pay-bursts-only-once property [8]. This paper is the first to consider arbitrary HSDF graphs in combination with non-starvation-free schedulers.

The analysis flow uses, similar to [5], the enabling jitter of tasks combined with their period to compute the response times of the tasks. The proposed flow computes this enabling jitter, in contrast to other methods, by using the best-case schedule and the worst-case schedule. The convergence of the analysis flow is guaranteed because the response times are monotonic in the enabling jitters of the tasks and because of the temporal monotonicity of dataflow graphs [20]. The analysis flow finishes when the enabling

M-SCOPES '13 June 19-21, 2013, St. Goar, Germany.

jitters are converged or when a violation of the temporal constraints is detected.

The presented analysis flow has a worst-case exponential time-complexity. However, we also provide a conservative (typically more pessimistic) version of the analysis flow which has a polynomial time-complexity. This conservative temporal analysis flow is based on linearization and can be used to speed up the analysis of complex applications but can lead to more pessimistic temporal results.

The outline of the paper is as follows. We first discuss related work in Section 2. The proposed temporal analysis flow is presented and detailed in Section 3. Next to that, Section 4 defines a conservative variant of this analysis flow which has a polynomial time-complexity. In section 5 we present a method to determine sufficient buffer sizes. The presented analysis flow is evaluated in Section 6 and Section 7 contains a case-study performed using the presented analysis flow. Section 8 illustrates directions for future work and we conclude with Section 9.

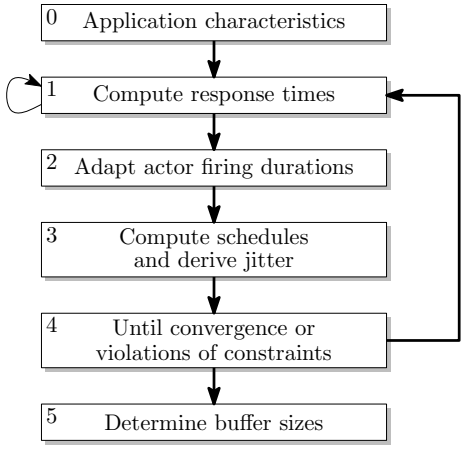


Figure 1: Overview of the analysis flow.

2. RELATED WORK

In [19] it has been shown that the effects of run-time schedulers that belong to the class of starvation-free schedulers can be included in HSDF analysis models. With these models the minimum throughput can be computed with a polynomial algorithm given only upper bounds on the execution time of the tasks. These models are also used to compute the minimum buffer capacities given a throughput constraint.

In this paper we show that also the effects of non-starvation-free schedulers can be analyzed using dataflow models. This allows to use similar algorithms for the computation of the minimum throughput and the minimum buffer capacities. This generalization comes at the cost of an exponential computational complexity of the analysis algorithms or a reduced accuracy. The accuracy can be improved by making use of knowledge about best-case execution times which is until now not considered for the analysis of run-time schedulers in combination with dataflow models.

An iterative procedure of traffic characterization and response time calculation [17, 13] is used by the SymTA/S approach [5]. However, this approach uses propagation of traffic instead of the computation of worst-case and best-case schedules. This can result, as we will show in Section 6, in a low accuracy because the applied traffic characterization does not capture the correlation between different streams accurately. The dataflow analysis techniques presented in this paper, do not suffer from this disadvantage because the schedules used to compute the enabling jitter, do capture the correlation of events. In [14] the SymTA/S approach is extended by calculating more accurate temporal end-to-end properties. However, it relies on similar event models as the original approach. Furthermore, these event models can not be determined for arbitrary cyclic graphs.

Real-time calculus for cyclic HSDF graphs [16] also makes use of a characterization of the traffic but correlation between streams is captured by a characterization in the time-domain instead of the time-interval domain. In the same paper a schedulability check given a throughput constraint is presented. However, it does only discuss (potentially cyclic) data dependencies in the application and does not include resource dependencies. Cyclic resource dependencies are considered in [6] but the combination with cyclic data dependencies is not considered. This combination is difficult because it requires an accurate translation between a traffic characterization in the time-domain and a traffic characterization in the time-interval domain.

We determine best- and worst-case schedules in the time-domain and calculate jitters based on these schedules. This jitter can be used to handle resource dependencies in the time-interval domain. The use of best- and worst-case schedules is possible thanks to the monotonicity property of dataflow graphs [20]. Thanks to this monotonicity we know that in every possible schedule, the finishes of tasks are later than the corresponding finishes in the best-case schedule and are earlier than the corresponding finishes in the worst-case schedule.

3. ANALYSIS FLOW

Figure 1 shows the proposed analysis flow. The flow can analyze task graphs which are specified by the application characteristics as discussed in Section 3.1. The temporal constraints of the application are included in this characterization by means of a periodic source.

Given these characteristics and an initial enabling jitter estimate (for example all enabling jitters equal to 0), the initial upper- and lower-bounds on the response times of tasks can be found using iterative fixed-point computation as is discussed in Section 3.2 (step 1).

The flow maintains a one-to-one relation between the given task graph and an HSDF model as we will discuss in Section 3.3. The calculated upper- and lower-bounds on the response times of tasks are used as the firing durations of the actors in this HSDF model (step 2). Based on the minimum and maximum firing durations of actors we calculate two periodic schedules, the first is a lower-bound on the best-case and the second an upperbound on the worst-case schedule of the dataflow model. These periodic schedules have the same period and can therefore be used to calculate the new enabling jitter of each task (step 3).

The process of response time computation, schedule computation and jitter determination (step 1, 2 and 3) is repeated until the upper- and lower-bound on the worst-case and best-case schedules respectively, remain unchanged and thus the jitter is converged or a violation of the temporal constrains is detected (step 4).

Finally, if enabling jitters are found, we can determine sufficient buffer sizes such that the temporal results, obtained in the previous steps of the analysis flow, are ensured (step

5). This step is discussed in Section 5.

Both the response time analysis step and the dataflow analysis step (schedule computation) can return infeasible which means that the temporal analysis concludes that the application with the given mapping and given constraints is not schedulable.

The complete analysis flow has a monotonic behavior. This means that a conservative temporal analysis result is found if the jitters have converged. The monotonic behavior of the flow is guaranteed by the fact that an iteration of the analysis flow can never lead to lower jitters. This is because the response time computation is monotonic in the jitter and because of monotonicity of dataflow graphs which is explained in Section 3.3.

3.1 Application Characteristics

The characteristics of an application consist of the application itself and the mapping of the application to a platform (task to processor specification). The application is described by one or more task graphs.

A task graph is a weakly connected directed graph of which the vertices represent tasks and the directed edges represent first-in first-out (FIFO) buffers. A task graph has one periodic source, τ_s, which executes strictly periodic with a period P_s. This strictly periodic execution of the source imposes the throughput constraint on the application. Note that a source is treated as a normal task with the difference that it is not activated by other tasks.

All the tasks in a task graph are activated by the source of that task graph. Therefore, task τ_i of a task graph with source τ_s executes, on average, periodically with a period equal to $P_i = P_s$. We require that no task, except the source, can execute before the corresponding source starts releasing data. To simplify the notation, we also define that the source of the task graph starts at time 0.

Tasks communicate using FIFO buffers which are directed from one task to an other. We use c_{ij} for a FIFO buffer from task τ_i to τ_j. FIFO buffers consists of an optional predefined capacity of φ containers which contain the data. The capacity of the FIFO buffers that are not fixed can be determined using buffer sizing techniques which will be presented in Section 5. The containers of a FIFO buffer can either be filled with data or be empty. We use $\bar{\varphi}_{ij}$ denote the initially full containers in FIFO c_{ij} and $\mathring{\varphi}_{ij}$ to denote the initially empty containers.

Every execution, a task τ_i acquires one full container from all FIFO buffers directed towards τ_i and releases one container, which τ_i fills with data, on all the output FIFO buffers. The execution of τ_i can only start if it can acquire an empty container from every output FIFO buffer. Furthermore, τ_i always releases the full container, acquired from an input FIFO buffer, as an empty container in the same buffer when the data is not needed anymore.

The minimum amount of execution time a task τ_i requires to complete an execution is specified by its Best-Case Execution Time (BCET), B_i and the maximum required execution time is specified by its Worst-Case Execution Time (WCET), C_i. The actual time it takes before a task finishes an execution after it gets enabled is specified by the response time of the task. Bounds on this response time are presented in the next section for two different run-time schedulers.

3.2 Response Times

The response time of a task is the time between the enabling of an execution and the corresponding finish of the execution. In this section we provide upper- and lower-bounds on the response time of tasks which have a jittered periodic enabling. These bounds will be used in the next sections to give guarantees on the temporal behavior of the application. Similar to [5], we define the bounds on the response time of a task τ_i using its period P_i and its enabling jitter J_i. The enabling jitter of a task defines the interval in which the task gets enabled.

Similar to [16, 14], only schedulers are supported for which the response times are monotonic in the traffic characterization. In our case the schedulers are required to have response times which are monotonic in the jitters of the tasks. If the response times are not monotonic in the jitters of the tasks, then the complete analysis flow is not monotonic and convergence of the analysis can not be guaranteed.

To find response time bounds, we use local scheduling analysis techniques [17, 13]. The basic idea is that the maximum/minimum interference of other tasks during a certain interval is added to the time the task takes to execute. For ease of understanding we use the most simple minimum response time definition, where the interference of other tasks is assumed to be 0. More accurate definitions of the minimum response time of a task are discussed in [13]. We define the minimum response time $\check{R}_i$ of a task τ_i as $\check{R}_i = B_i$.

As discussed in [13], the maximum number of activations of a task in a time interval Δt is equal to:

$$\hat{\eta}_j(\Delta t) = \left\lceil \frac{J_j + \Delta t}{P_j} \right\rceil \tag{1}$$

This maximum number of activations can be used to provide upper-bounds on the response time of a task. In the following two paragraphs we provide such an upper-bound for tasks that are scheduled with a static priority pre-emptive scheduler and with a round-robin scheduler respectively.

3.2.1 Static Priority Pre-emptive Scheduling

The maximum response time of a task, τ_i, scheduled using a static priority pre-emptive scheduler can be determined using the following method [17].

$$w_i(q) = q \cdot C_i + \sum_{j \in hp(i)} \hat{\eta}_j(w_i(q)) \cdot C_j \tag{2}$$

$$\hat{R}_i = \max_{1 \leq q} \left(w_i(q) - (q-1) \cdot P_i \right) \tag{3}$$

Equation 2 calculates $w_i(q)$ which is the maximum amount of time it takes to finish q executions of a task τ_i after it is enabled. We use $hp(i)$ as the set of tasks running on the same processor as τ_i and which have a higher priority than τ_i. As is shown in [17], iterative fixed-point computation can be used to compute $w_i(q)$. Equation 3 calculates the maximum response time using $w_i(q)$. Only values of q for which $w_i(q) \geq q \cdot P_i$ holds need to be considered [17]. As long as $w_i(q) \geq q \cdot P_i$ holds, no lower priority task is executed. We define this as task τ_i being in a consecutive execution.

3.2.2 Round-Robin Scheduling

A round-robin scheduler activates the tasks one after each other and does not pre-empt the execution of a task. A round-robin scheduler is thus starvation-free because the interference of other tasks can be bounded when only the execution times of the other tasks are known. However, the accuracy of the temporal analysis results, of a system with a round-robin scheduler, can be improved by taking the actual traffic of the application into account. The method

presented in this paper can therefore be used to improve these temporal results.

A round-robin scheduler ensures that during q executions of a task τ_i, maximally q executions of each of the other tasks can occur. With this extra information we can refine the maximum number of activations of a task τ_j in an interval Δt given maximally q activations to:

$$\hat{\eta}'_j(q, \Delta t) = \min\left(q,\ \hat{\eta}_j(\Delta t)\right) \tag{4}$$

The maximum response time of task τ_i given round-robin scheduling can now, similarly as for static priority pre-emptive scheduling, be calculated with Equations 5 and 6 where $T(i)$ is the set of tasks running on the same processor as task τ_i without task τ_i itself. Again iterative fixed-point computation is used to compute $w_i(q)$ and similar to the method for static priority pre-emptive scheduling, only values of q for which $w_i(q) \geq q \cdot P_i$ holds need to be considered.

$$w_i(q) = q \cdot C_i + \sum_{j \in T(i)} \hat{\eta}'_j(q, w_i(q)) \cdot C_j \tag{5}$$

$$\hat{R}_i = \max_q \left(w_i(q) - (q-1) \cdot P_i\right) \tag{6}$$

3.3 Temporal Analysis

This section provides a method to calculate the enabling jitter of tasks based on dataflow analysis techniques. We use a dataflow model for which we show that it is a temporally conservative abstraction of the task graphs of the application.

3.3.1 Analysis Model

The used dataflow model is an HSDF graph. An HSDF graph is a directed graph $G = (V, E, \delta, \rho)$ that consists of a set of actors V and a set of directed edges E between those actors. Actors communicate by producing and consuming tokens over the edges, which represent unbounded queues. An edge e_{ij} initially contains $\delta(e_{ij})$ tokens. An actor is enabled to fire if a token is available on each of its input edges. Furthermore, the firing duration, ρ_i, specifies the difference between the finish time and the start time of a firing of actor v_i. At the start of a firing, the actor consumes one token from all the input edges and when it finishes, it produces atomically one token in each of the output edges.

Functionally deterministic dataflow graphs, such as an HSDF graph, have a monotonic temporal behavior [20]. This has a number of implications which are used in the next sections. We use the fact that increasing/decreasing the firing duration of one actor in the graph can never lead to an earlier/later enabling of each of the actors in the graph. And similarly, increasing/decreasing the amount of initial tokens in the graph can never lead to a later/earlier enabling of each of the actors in the graph.

Figure 2 illustrates the relation between the dataflow model and the task graph. A task τ_i in the task graph corresponds with a unique actor v_i in the dataflow model. A FIFO buffer c_{ij} between tasks τ_i and τ_j is modeled using two oppositely directed edges between actors v_i and v_j. The forward edge, e_{ij}, models the flow of full containers and contains initially $\bar{\varphi}_{ij}$ tokens. Similarly, the backwards edge, e_{ji}, models the flow of empty containers and has initially $\mathring{\varphi}_{ij}$ tokens. If the maximum size of a FIFO buffer from task τ_i to τ_j is not fixed, the number of tokens on the backwards edge, e_{ji}, is assumed to be infinite. This means that the backwards edge does not influence the enabling of the producing actor (v_i). In Section 5 we present a method with

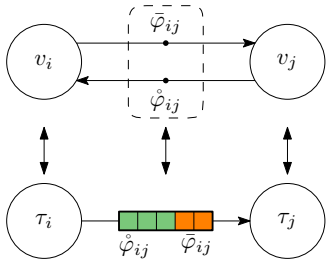


Figure 2: One-to-one relation between dataflow model and task graph.

which a sufficient size can be determined for buffers which do not have a fixed maximum size.

A dataflow model is temporally conservative to a task graph if the worst-case temporal behavior of the task graph is captured conservatively by the dataflow model. This can be shown by proving that all of the container arrival times are bounded from above by the arrival times of the corresponding tokens [20]. For each pair of task and corresponding actor we have to prove:

$$\forall_i :\ e(i) \leq \hat{e}(i) \implies \forall_i :\ f(i) \leq \hat{f}(i) \tag{7}$$

Where $e(i)$ is the enabling time of execution i of a task and $\hat{e}(i)$ the enabling of the corresponding actor firing. The finish time of execution i of the task is equal to $f(i)$ and $\hat{f}(i)$ corresponds with the finish time of the corresponding actor firing. In [20] it is shown that if (7) holds for every task and its corresponding actor, the dataflow model is temporally conservative to the task graph.

If we want to bound the best-case behavior of the task graph we have to show something similar. We then have to prove:

$$\forall_i :\ e(i) \geq \check{e}(i) \implies \forall_i :\ f(i) \geq \check{f}(i) \tag{8}$$

3.3.2 Determining Enabling Jitter

With the determined worst-case and best-case response times of tasks, we define two instances of the dataflow model to determine new enabling jitters of the tasks. Periodic schedules are determined with both the instances of the dataflow model. The first dataflow model bounds the worst-case behavior from above and the other dataflow model bounds the best-case behavior from below.

The difference between the best-case and worst-case dataflow model is the firing durations of the actors. In the best-case dataflow model, the firing duration of actor v_i is equal to $\check{\rho}_i = \check{R}_i$, with $\check{R}_i$ the minimum response time of the corresponding task τ_i. The firing duration of the same actor in the worst-case dataflow model is equal to $\hat{\rho}_i = \hat{R}_i$ with $\hat{R}_i$ the maximum response time of τ_i.

We first calculate a worst-case periodic schedule using the worst-case dataflow model. This periodic schedule bounds the self-timed schedule from above. In such a self-timed schedule of a dataflow model, each actor starts its firing as soon as it is enabled, . For HSDF graphs, a periodic schedule is rate-optimal [11]. We use Algorithm 1 to calculate the earliest start time of each actor such that it only fires after it gets enabled on each edge and such that it can fire periodically. Algorithm 1 is a Linear Programming (LP) algorithm which has a polynomial time-complexity.

All the tasks in the application can not execute before the source in the application is started. Therefore all actors in

Algorithm 1

Minimize

$$\sum_{v_i \in V} \hat{s}_i$$

Subject to

$$\hat{s}_s = 0 \tag{9}$$

$$\forall_{e_{ij} \in E} \ : \ \hat{s}_j - \hat{s}_i \geq \hat{\rho}_i - \delta(e_{ij}) \cdot P_i \tag{10}$$

the corresponding dataflow model are only enabled after the actor corresponding to the source starts producing tokens. We start the source at time 0 (see Equation 9) and we thus also know that the start time of each actor is larger or equal than 0. Note that we use $\hat{s}_s$ for the start time of the source.

The schedule needs to be determined such that it is admissible. A schedule is admissible when all the tokens are consumed from an edge after they are produced on that edge. This can be ensured by adhering to the precedence constraint imposed by each edge e_{ij} [11]:

$$\forall_{k \in \mathbb{N}} : \ \hat{s}_j(k) \geq \hat{s}_i(k - \delta(e_{ij})) + \hat{\rho}_i \tag{11}$$

We use $\hat{s}_i(k)$ as the start time of firing k of actor v_i. The schedule that is determined is periodic with a period P_i for actor v_i and the start time of the first firing is denoted by $\hat{s}_i$. The start time of of firing k of v_i is therefore equal to:

$$\hat{s}_i(k) = \hat{s}_i + k \cdot P_i \tag{12}$$

Furthermore, for every edge e_{ij} it holds that $P_i = P_j$. In a periodic schedule we can therefore adhere to (11) by ensuring:

$$\forall_{k \in \mathbb{N}} : \ \hat{s}_j + k \cdot P_i \geq \hat{s}_i + (k - \delta(e_{ij})) \cdot P_i + \hat{\rho}_i \tag{13}$$

Equation 13 can be rewritten to Equation 10 which means that Algorithm 1 determines an admissible periodic schedule. If Algorithm 1 returns infeasible, a schedule with the required throughput does not exist, i.e., the period is too small or the number of tokens in the graph is not sufficient.

The best-case periodic schedule is determined using the best-case dataflow model. With this best-case dataflow model we calculate the earliest possible moment at which an actor is enabled for its first firing. We then assume a periodic schedule with period P_i for each actor v_i starting at this earliest enabling time. We calculate this earliest enabling time of the first firings by only considering edges in the dataflow model that initially do not contain tokens. An edge e_{ij} on which initial tokens are available does not impose a precedence constraint on the first firing of actor v_i which means that we do not have to take e_{ij} into account when determining the best-case periodic schedule. This is equal to assuming an infinite number of initial tokens on these edges. We use algorithm 2 to determine the earliest possible enabling of the first firing of each actor.

Note that although we do not take the edges with tokens into account, the actors are still being enabled by the source. This is thanks to the fact that there is always a path of edges without tokens to each actor, otherwise the actor was enabled before the source actor starts firing. Because actors are still enabled by the source, the fastest enabling is a periodic enabling that starts with the earliest enabling of the first firing of an actor.

Both Algorithm 1 and 2 can be written as a dual of the uncapacitated network flow problem which can be solved efficiently [1]. In fact it can even be written as a shortest

Algorithm 2

Minimize

$$\sum_{v_i \in V} \check{s}_i$$

Subject to

$$\check{s}_s = 0 \tag{14}$$

$$\forall_{e_{ij} \in E'} \ : \ \check{s}_j - \check{s}_i \geq \check{\rho}_i \tag{15}$$

$$\text{with } E' = \{e \mid e \in E \ \wedge \ \delta(e) = 0\}$$

path problem that can be solved using the Bellman-Ford algorithm. The Bellman-Ford algorithm can detect negative cycles and the detection of such a negative cycle indicates infeasibility of the specified temporal constraints.

The interval in which a task can get enabled is bounded by the two determined periodic schedules. Both these periodic schedules have the same period which means that the difference between the start of a firing in these schedules is equal for every firing. The new enabling jitter of task τ_i corresponding to actor v_i can thus be calculated using the difference between the start times of these schedules:

$$J_i = \hat{s}_i - \check{s}_i \tag{16}$$

3.3.3 Proof of Conservativeness

In this section we proof that the temporal behavior of the application is modeled conservatively with the analysis models. This also proofs that the determined jitter is conservative to the actual jitter of the the tasks in the application.

Task τ_i has a schedule with period P_i and maximum deviation J_i from the periodic enabling. Furthermore, $\check{s}_i$ is a lower-bound on the earliest possible enabling time of the first execution of task τ_i. The enabling time of an execution k of task τ_i can thus be bounded by the interval:

$$e_i(k) = \check{s}_i + k \cdot P_i + J_i(k) \qquad \text{with } 0 \leq J_i(k) \leq J_i \tag{17}$$

This enabling is thus bounded from above with the following equation. Note that $\hat{e}_i(k)$ is the enabling time of actor v_i in the worst-case schedule of the dataflow model.

$$\begin{aligned} e_i(k) \leq \hat{e}_i(k) &= \check{s}_i + J_i + k \cdot P_i \\ &= \hat{s}_i + k \cdot P_i \end{aligned} \tag{18}$$

Given the semantics of dataflow models we know that the finish time of firing k of actor v_i is equal to:

$$\begin{aligned} \hat{f}_i(k) &= \hat{e}_i(k) + \hat{\rho}_i \\ &= \hat{e}_i(k) + \hat{R}_i \end{aligned} \tag{19}$$

Furthermore, we know from the definition of the response time (see Section 3.2) the following:

$$\forall_q : \hat{R}_i \geq w_i(q) - (q-1) \cdot P_i \qquad \text{and thus:}$$

$$\forall_q : w_i(q) \leq \hat{R}_i + (q-1) \cdot P_i$$

We know that $w_i(q)$ is an upperbound on the amount of time it takes to finish q executions of task τ_i after it is enabled. We therefore can bound the finish time of each execution k of task τ_i als follows:

$$f_i(k) \leq \max_q \left(e_i \left(k - (q-1) \right) + w_i(q) \right) \tag{20}$$

By using the definition of the response time and the defi-

nition of $\hat{e}_i(k)$ we can rewrite this bound to:

$$\begin{aligned} f_i(k) \leq & \max_q \left(\hat{e}_i \left(k - (q-1)\right) + \hat{R}_i + (q-1) \cdot P_i \right) \\ \leq & \max_q \left(\hat{s}_i + (k - (q-1)) \cdot P_i + \hat{R}_i + (q-1) \cdot P_i \right) \\ \leq & \max_q \left(\hat{s}_i + k \cdot P_i + \hat{R}_i \right) \\ \leq & \ \hat{e}_i(k) + \hat{R}_i \end{aligned} \tag{21}$$

From Equations 18, 19 and 21 we conclude that Equation 7 holds. This means that the provided dataflow model captures the worst-case temporal behavior of the application conservatively.

Equation 8 can be proven similarly by using the definition of the minimum response time of tasks.

4. LINEARIZED RESPONSE TIME

In this section we provide a temporally conservative abstraction of the steps 1, 2, 3 and 4 of the analysis flow of Figure 1. This conservative analysis flow has a polynomial time-complexity which is accomplished by linearizing the response times. The computation of these linearized response times does not require iterative fixed point computation. The linearized response time of a task is linear in its enabling jitter which means that we can combine it with Algorithms 1 and 2 to obtain an algorithm that calculates the complete flow using an LP algorithm which has a polynomial time-complexity.

In this paper we provide this linearization for a system with a static priority pre-emptive scheduler:

$$\hat{R}_i = \max_q \left(w_i(q) - (q-1) \cdot P_i \right) \text{ with} \tag{22}$$

$$w_i(q) = q \cdot C_i + \sum_{j \in hp(i)} \left\lceil \frac{J_j + w_i(q)}{P_j} \right\rceil \cdot C_j \tag{23}$$

We first define an upperbound $\tilde{w}_i(q)$ on $w_i(q)$ by bounding $\lceil x \rceil$ with $(x+1)$:

$$\forall_{q \in \mathbb{N}} \ w_i(q) \leq \tilde{w}_i(q) \qquad \text{with}$$
$$\tilde{w}_i(q) = q \cdot C_i + \sum_{j \in hp(i)} \left(\frac{J_j + \tilde{w}_i(q)}{P_j} + 1 \right) \cdot C_j \tag{24}$$

This upperbound $\tilde{w}_i(q)$ can be written as an equation linear in q and linear in the enabling jitters of the tasks. We use $\alpha_i = \sum_{m \in hp(i)} \frac{C_m}{P_m}$ as a shorthand notation.

$$\begin{aligned} \tilde{w}_i(q) &= q \cdot C_i + \sum_{j \in hp(i)} \left(\frac{J_j}{P_j} + 1 \right) \cdot C_j + \tilde{w}_i(q) \cdot \alpha_i \\ \tilde{w}_i(q) \cdot (1 - \alpha_i) &= q \cdot C_i + \sum_{j \in hp(i)} \left(\frac{J_j}{P_j} + 1 \right) \cdot C_j \\ \tilde{w}_i(q) &= \frac{q \cdot C_i + \sum_{j \in hp(i)} \left(\frac{J_j}{P_j} + 1 \right) \cdot C_j}{1 - \alpha_i} \\ \tilde{w}_i(q) &= q \cdot \frac{C_i}{1 - \alpha_i} + \sum_{j \in hp(i)} \frac{J_j \cdot \frac{C_j}{P_j} + C_j}{1 - \alpha_i} \end{aligned} \tag{25}$$

We can substitute this upperbound $\tilde{w}_i(q)$ in Equation 22 to define and upperbound $\tilde{R}_i$ on the maximum response time of task τ_i:

$$\begin{aligned} \tilde{R}_i &= \max_q \left(\tilde{w}_i(q) - (q-1) \cdot P_i \right) \\ \tilde{R}_i &= \max_q \left(q \cdot \frac{C_i}{1 - \alpha_i} + \sum_{j \in hp(i)} \frac{J_j \cdot \frac{C_j}{P_j} + C_j}{1 - \alpha_i} - (q-1) \cdot P_i \right) \\ \tilde{R}_i &= \max_q \left(q \cdot \left(\frac{C_i}{1 - \alpha_i} - P_i \right) + \sum_{j \in hp(i)} \frac{J_j \cdot \frac{C_j}{P_j} + C_j}{1 - \alpha_i} + P_i \right) \end{aligned} \tag{26}$$

The equation inside the max expression of (26) is strictly increasing in q if:

$$\frac{C_i}{1 - \alpha_i} > P_i \tag{27}$$

If Equation 27 holds, $\tilde{R}_i$ thus would become infinite. This means that τ_i is not schedulable. If Equation 27 does not hold, the equation inside the max expression of (26) is non-increasing which means that the response time is finite and can be found by choosing q equal to 1. In fact the upperbound on the response time of τ_i is in that case equal to $\tilde{w}_i(1)$ and no iterative fixed-point computation is required to determine it. This removes the inner iteration loop the analysis flow as defined in Figure 1. The upperbound on the response time which we use in the remainder of this section, is linear in the enabling jitters of the task and is defined as follows.

$$\tilde{R}_i = \sum_{j \in hp(i)} J_j \cdot \frac{\frac{C_j}{P_j}}{1 - \alpha_i} + \frac{C_i + \sum_{j \in hp(i)} C_j}{1 - \alpha_i} \tag{28}$$

To determine if a task set is schedulable, it is sufficient to verify whether Equation 27 does not hold for every lowest priority task on each processor. If the task set is schedulable we can use the upperbounds on the response times of tasks to define an analysis algorithm with a polynomial time complexity. We use $\tilde{R}_i$ as defined in Equation 28 as the maximum response time of task τ_i and B_i as the minimum response time of task τ_i.

Similarly as the method described in Section 3.3 we use two instances of the dataflow model of the application, a best-case dataflow model and a worst-case dataflow model. We use the minimum/maximum response time of task τ_i as the firing duration of actor v_i in the best/worst-case schedule respectively.

We again use an LP algorithm to calculate the start times of the best- and worst-case schedule. The chosen minimum response time of a task does not dependent on the jitters of tasks. We can therefore use Algorithm 2 to compute the minimum start times of the tasks which are constant throughout the remainder of this section. If a minimum response time is used that is dependent on the jitters of tasks we could have linearized it similarly as the maximum response time and we could include the determination of the minimum schedule in the algorithm below. However, this is out of the scope for this paper.

We use a similar algorithm as Algorithm 1 to calculate the maximum start time of tasks. However, we use the fact that $\tilde{R}_i$ is linear in the jitters of the tasks. Because of this linearity we can include the jitters in the LP algorithm which makes the outer iteration loop of the analysis flow, as defined in Figure 1, redundant.

Algorithm 3 is used to compute the start times of upperbound on the worst-case schedule. Compared to Algo-

rithm 1, constraint (10) is changed to constraint (30) by using Equation 28 and moving the part of the equation that is dependent on the jitter to the left side of the constraint. Furthermore, constraint (31) is added. Note that the helper functions, β and γ, do not contain any variable and thus are constant.

Algorithm 3

Minimize

$$\sum_{v_i \in V} \hat{s}_i$$

Subject to

$$\check{s}_s = 0 \ \wedge \ \hat{s}_s = 0 \tag{29}$$

$$\forall_{e_{ij} \in E} : \hat{s}_j - \hat{s}_i - \sum_{k \in hp(i)} J_k \cdot \beta(i,k) \geq \gamma(i) - \delta(e_{ij}) \cdot P_i \tag{30}$$

$$\forall_{v_i \in V} : J_i = \hat{s}_i - \check{s}_i \tag{31}$$

With $\beta(i,k) = \frac{\frac{C_k}{P_k}}{1-\alpha_i}$ and $\gamma(i) = \frac{C_i + \sum_{k \in hp(i)} C_k}{1-\alpha_i}$

5. BUFFER SIZING

In this section we provide methods to determine the sizes of the buffers such that the temporal constraints can be met. Existing buffer sizing algorithms for dataflow graphs, such as [12], can not be applied because they choose arbitrary start times for the actors such the buffers are minimal. These arbitrary start times can lead to a larger jitter than the jitter calculated in steps 1, 2, 3 and 4 of the analysis flow. This is obviously not allowed.

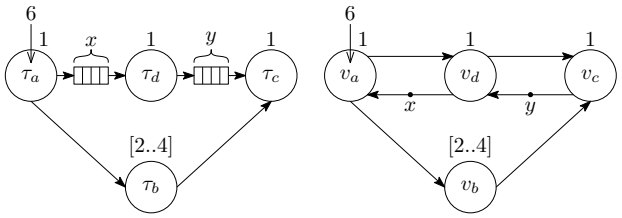
We first present a method to determine sufficient buffer sizes. This method chooses the buffers such that the periodic schedules as defined in step 3 in the analysis flow remain admissible. Second, we present a buffer minimization algorithm similar to the algorithm of [12]. This algorithm is possible because we base it on the linearized version of the analysis flow as presented in Section 4.

5.1 Sufficient Buffer Sizes

Because of monotonicity of HSDF graphs, we know that a larger number of tokens can never lead to later enabling times of actors. This holds also the other way around; a smaller number of tokens can never lead to earlier enabling times of actors. The lower-bound on the best-case schedule as defined in Section 3.3 basically assumes an infinite number of tokens on each edge that may contain initial tokens. Lowering this number of initial tokens can never lead to earlier enabling times. This means that the determined periodic lower-bound is still a lower-bound on the best-case scheduling of the HSDF model after choosing a number of initial tokens.

Therefore, for each edge e_{ij} the number of sufficient tokens $\delta(e_{ij})$ is sufficient if it does not delay the periodic upper-bound on the worst-case schedule of the dataflow model. This can be ensured by determining the amount of tokens such that the maximum start times as calculated with Algorithm 1 are feasible. In fact we have to ensure that Equation 10 holds for every edge e_{ij}. We use the start times calculated with Algorithm 1 and ensure that Equation 10 holds by choosing $\delta(e_{ij})$ equal to the smallest integer that satisfies:

$$\delta(e_{ij}) \geq \frac{\hat{\rho}_i + \hat{s}_i - \hat{s}_j}{P_i} \tag{32}$$



(a) task graph (b) dataflow graph

Figure 3: Buffer minimization example.

5.2 Buffer Minimization

The linearized analysis flow as presented in Section 4 can be used to calculate minimum sufficient buffer capacities. We again choose the best-case schedule independent of the jitter of the tasks by only using the BCET of tasks. The earliest start times of the first firings can be calculated using Algorithm 2. Algorithm 4 is used to calculate minimum sizes for the buffers which do ensure that a periodic schedule exists.

Minimizing the amount of tokens does not lead to the smallest possible maximum start times and does also not lead to minimum jitters. However, Algorithm 4 takes the effect that increasing jitters has on the maximum response time of tasks, into account and uses the available freedom to find a minimum amount of tokens that ensures the temporal constraints.

The number of tokens $\delta(e)$, should be integer. If this is enforced in the algorithm, Algorithm 4 becomes an Integer Linear Programming (ILP) algorithm which has a worst-case non-polynomial time-complexity. Thanks to the monotonicity property of dataflow graphs, sufficient buffer sizes can be found by keeping $\delta(e)$ a real-value and by rounding $\delta(e)$ up to nearest integer value after the execution of the algorithm finishes. This potentially leads to pessimistic buffer sizes however they can be determined with a polynomial time-complexity.

Algorithm 4

Minimize

$$\sum_{e_{ij} \in E} \delta(e_{ij})$$

Subject to

$$\check{s}_s = 0 \ \wedge \ \hat{s}_s = 0 \tag{33}$$

$$\forall_{e_{ij} \in E} : \hat{s}_j - \hat{s}_i - \sum_{k \in hp(i)} J_k \cdot \beta(i,k) \geq \gamma(i) - \delta(e_{ij}) \cdot P_i \tag{34}$$

$$\forall_{v_i \in V} : J_i = \hat{s}_i - \check{s}_i \tag{35}$$

With $\beta(i,k) = \frac{\frac{C_k}{P_k}}{1-\alpha_i}$ and $\gamma(i) = \frac{C_i + \sum_{k \in hp(i)} C_k}{1-\alpha_i}$

Example 1:
Figure 3a shows the task graph of an application which consists of four tasks. The application is executed on two processors which both have a static priority pre-emptive scheduler. The first processor executes tasks τ_a (with the highest priority) and τ_d (lowest priority). The other processor executes the other two tasks. Task τ_c has the highest priority on this processor and task τ_d the lowest priority. Tasks τ_a, τ_c and τ_d have a constant execution time of 1 time unit and task τ_b has an execution time between 2 and 4 time units.

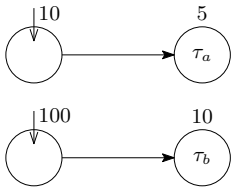


Figure 4: Two streams scheduled with a round robin scheduler.

The objective is to find minimum buffer sizes for the buffers c_{ad} (size of x locations) and c_{dc} (y locations) which guarantee the throughput constraint imposed by the period of 6 *time units of task τ_a.*

The dataflow graph that models the temporal behavior of the application is shown in Figure 3b. The number of required initial tokens on edge e_{da} (denoted with x) corresponds with the required size of c_{ad}. The number of initial tokens on e_{cd} (denoted with y) corresponds with the size of c_{dc} We first use this dataflow model with the analysis method presented in Section 3 to find bounds on the possible schedules of the tasks. After convergence of the analysis flow we find the following start times, jitters and response times:

	v_a	v_b	v_c	v_d
$\check{s}$	0	1	3	1
$\hat{s}$	0	1	7	1
J	0	0	4	0
$\hat{R}$	1	6	1	2

Using these start times and response times we can compute sufficient buffer sizes with Equation 32: $x = 1$ and $y = 2$.

When we use the method of section 5.2 we compute the following start times, jitters and response times:

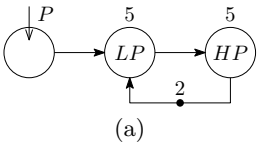
	v_a	v_b	v_c	v_d
$\check{s}$	0	1	3	1
$\hat{s}$	0	1	8	3
J	0	0	5	2
$\hat{R}$	1	7	1	$\frac{12}{5}$

With Algorithm 4 we compute that a size of 1 *location is sufficient for both the buffers c_{ad} and c_{dc} ($x = 1$ and $y = 1$). Despite of the fact that the jitters and worst-case response times are over-approximated, the computed buffer sizes are less. Note that the algorithm uses the freedom in the worst-case start time of actor v_d to find smaller buffers.*

6. EVALUATION

In this section we evaluate the presented analysis approach. We give four examples of the use of the analysis approach and show some of the consequences regarding the differences with other temporal analysis methods.

The first example is shown in Figure 4. It shows a snippet of an application which contains two tasks, τ_a and τ_b. Both these tasks process a periodic stream of data with a different period (10 and 100 time units respectively). The tasks share a resource which is scheduled using a round-robin scheduler. The WCET of each task is shown above the vertex. Traditional dataflow analysis would use a response time for a task equal to the sum of the WCETs. Using this analysis,



(a)

P	25	20	15	14
R_{LP}	10	10	10	15
$R_{LP}(l)$	15	17.5	25	X

(b)

Figure 5: Example of a topology for which linearization of the response time can become problematic (a) and corresponding normal and linearized maximum response times of the low-priority task (b).

the actor corresponding to a task can only fire once per response time. The response time of τ_a is equal to 15 time units which means that its throughput constraint (execute once per 10 time units) cannot be met.

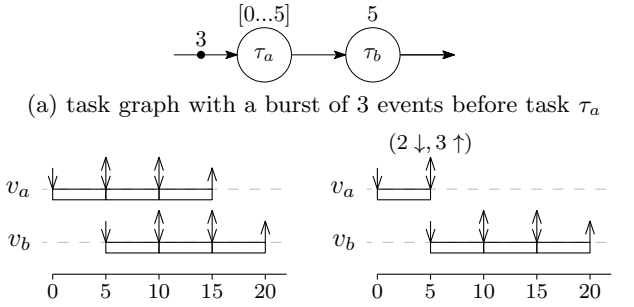
If we use the analysis provided in this paper, we will find maximum response times for both the tasks equal to 15 time units. The analysis finds out that the interference of τ_b on τ_a is sporadic and that on average τ_a can execute with a period of 10 time units. The throughput constraints can thus be met.

The second example considers the linearization approach presented in this paper. This linearization can be used to speed up the temporal analysis of applications. However, it over-approximates jitter which can in some cases result in very conservative results. This is because jitter is also used in the response time of a task and an increase in response time can again lead to an increase of the jitter.

Figure 5a illustrates this issue. The interference from the high-priority task (HP) on the low-priority task (LP) is over-approximated when linearization is used. Figure 5b shows the response times of the LP task for different periods of the source. Decreasing the period leads to a larger difference between the normal response time (R_{LP}) and the linearized response time ($R_{LP}(l)$) and eventually leads to an unnecessary conclusion of infeasibility. Note that this effect does not occur in the case of starvation-free schedulers. This is because when such a scheduler is used, the response time of tasks can be defined independent of the execution rate of the other tasks [20]. Therefore, no circular resource dependencies between tasks have to be taken into account which prevents high inaccuracies in case of linearized response times.

The next example considers bursts in the application. Such burst can lead to inaccurate temporal analysis results. The end-to-end latency is in fact often computed using a notion of response time that includes the sum of the times that data resides in the queues, plus the time it takes to process the data. Because in a path, often a burst can happen at the input of each task, the local worst-case response time of each of the tasks in the path becomes large due to the worst-case queuing time in each queue. However, in case of a burst, the same event that gets processed along a path does not experience this delay at each task.

The temporal analysis method we propose uses dataflow analysis techniques. Therefore, the correlation of events is preserved and the time that data resides in the buffer does not need to be taken into account in the response time of a task. The end-to-end latency can be calculated using the schedules of the dataflow graph which use the actual enabling of an actor. This means that we do not accumulate times that data resides in the different buffers. Only the buffer time at the beginning of the path needs to be taken



(a) task graph with a burst of 3 events before task τ_a

(b) worst-case schedule (c) schedule with burst on e_{ab}

Figure 6: Illustration of burst in an application.

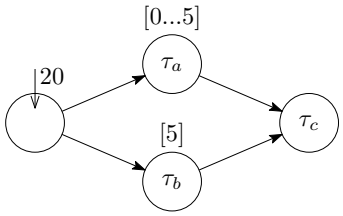


Figure 7: Example of inaccuracy in the temporal analysis if the correlation between c_{ac} and c_{bc} is lost.

into account. Our analysis method thus has the pay-bursts-only-once property.

An example of burst in an application is illustrated in Figure 6. The graph consists of two tasks, τ_a and τ_b. The incoming queue of task τ_a contains 3 tokens to indicate that a burst of 3 events occurred. Task τ_a has a response time between 0 and 5 time units and task τ_b has a constant response time equal to 5 time units. The worst-case schedule of this task graph is shown in Figure 6b. The downwards arrows indicate the start of executions and the finish of an execution is indicated with a upwards arrow. A simultaneous start and finish of two executions is depicted by a bidirectional arrow. In the schedule of Figure 6c the second and third executions of task τ_a are assumed to take 0 time units, equal to the minimum response time of the task. The tuple above the arrow at time 5 indicates the amount of executions that are started and finished respectively at time 5. In this schedule, a burst of 3 events happens at c_{ab} because 3 tokens are produced at time 5. However, despite of this burst at c_{ab}, the worst-case production times of τ_b are in Figure 6c equal to the production times in Figure 6b. The worst-case latency from begin to end is thus not influenced by the presence of burst behavior on c_{ab}. In fact, the worst-case latency of a path can be determined using the worst-case schedule only. This is because of monotonicity of dataflow graphs which are used during the analysis. This monotonicity tells us that producing tokens earlier can never lead to any later finish time of any actor in the graph.

The fourth example is shown in Figure 7. This figure shows an example of a topology for which the temporal analysis suffers from inaccuracies if the correlation between streams is lost. The example has three tasks and a periodic source. Task τ_a has a response time between 0 and 5 time units and task τ_b has a constant response time equal to 5 time units. With our analysis flow we conclude that task τ_c has a minimum start time equal to the maximum start time. The enabling jitter of task τ_c is thus equal to 0. Methods

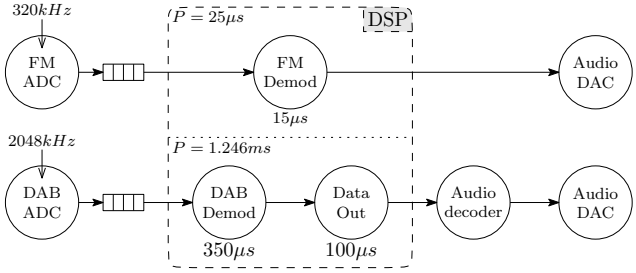


Figure 8: Example of an FM and a DAB application sharing one DSP.

which can not correlate streams use the maximum of the arrival jitters on the different incoming edges as the enabling jitter [5]. In this case they would conclude that the enabling jitter of task τ_c is equal to 5 which is inaccurate.

7. CASE STUDY

Figure 8 shows the task graphs of two applications, a Frequency Modulation (FM) receiver and a Digital Audio Broadcasting (DAB) receiver. Both applications share one DSP core which executes both demodulation stages. This DSP uses a round-robin scheduler to schedule the tasks at run-time. For simplicity we always schedule the DAB demodulation task and the Data Out task together. For clarity we only present the demodulation stage of the application.

The sample rate of the Analog-to-Digital Converters (ADCs) differ in both the application because the bandwidth requirements are different (2048kHz sample rate for DAB and 320kHz for FM). Furthermore, the demodulation task in the DAB application performs an FFT of size 2048 which means that the DAB demodulation task uses blocks of 2048 samples while the FM demodulation uses much smaller blocks of 8 samples.

These differences in sample rate and block size result in a different enabling rate of the two demodulation tasks. The FM demodulation task gets enabled with a period of $\frac{\text{blocksize}}{320kHz} = 25\mu s$. The DAB demodulation tasks is enabled with a period of $1.246ms$ which includes the symbol period of $1ms$ plus the guard interval of $0.246ms$ for DAB Transmission Mode I [3]. The execution times of the tasks are assumed to be constant. The FM demodulation takes $15\mu s$ and DAB demodulation and Data Out task together take $450\mu s$.

We now want to verify whether both applications meet their throughput constraint and how much buffering is required between the ADCs and the demodulation tasks. Traditional methods for analyzing systems with round-robin schedulers use the fact that for one execution of a task, the task has to wait for at most one execution of all the other tasks. However because no traffic characterization is taken into account, they assume that this interference of other tasks happens during all executions. In that case, one would conclude that the FM demodulation can only execute once per $465\mu s$ which means that it can not meet its throughput constraint.

The fix for this problem that is currently used, is executing the FM demodulation task 50 times in a row. 50 Executions of the FM demodulation task are grouped together which means that the DAB demodulation task interferes at most once per 50 executions. The 50 executions take $750\mu s$ and have a period of $1250\mu s$. With this grouping the traditional

analysis method concludes that both the applications can meet their throughput. The response time of both the demodulation tasks is equal to $1.2ms$. The FM demodulation can start after the ADC has produced $50 \cdot 8$ samples, $1.25ms$ after the source starts. Using the method of Section 5 we conclude that a buffer size of $2 \cdot 50$ times the blocksize is sufficient for the buffer between the FM ADC and the FM demodulation task ($2 \cdot 50 \cdot 8$ samples). The DAB demodulation starts after the ADC has produced 2048 samples, $1ms$ after the sources starts. Two times the blocksize ($2 \cdot 2048$ samples) is thus sufficient for the buffer between the DAB ADC and the DAB demodulation task.

The problem can be solved more accurate and elegant with the analysis method that is described in this paper. With this analysis method it is not needed to group the executions of the FM demodulation task together because the actual traffic/enablings are taken into account. The analysis methods described in Section 3 conclude that the throughput constraints can be met. The worst-case response times of both the demodulation tasks are equal to $465\mu s$ which is much less than the response times of the grouping solution (61.25% less). The buffer sizes can be determined using the method presented in Section 5.1. The FM demodulation task can start $25\mu s$ after the source starts (8 samples) and using Equation 32 we conclude that the size of the buffer between the FM ADC and FM demodulation needs to be larger than $\frac{25+465}{25}$ times the blocksize. A buffer size of 20 times the blocksize ($20 \cdot 8$ sample) is thus sufficient for this buffer. This is a reduction of 80% compared to the solution of the previous paragraph. The buffer between the DAB ADC and the DAB demodulation task needs to be larger than $\frac{1000+465}{1246}$ times the blocksize. A size of 2 times the blocksize ($2 \cdot 2048$ samples) is thus sufficient for this buffer.

8. FUTURE WORK

The analysis method which we presented in this paper is the first method that combines the analysis of non-starvation-free schedulers with cyclic applications. We are aware that there are a lot of possibilities, to improve the accuracy of the analysis results, which we did not discuss in this paper to keep the method simple. For example, in [13] improvements on the bounds on response times of tasks are discussed. An example of such an improvement is by taking the minimum distance between enablings into account.

We also see this paper as a first step towards support for a broader class of dataflow graphs, like Synchronous Dataflow (SDF) graphs. Certain applications can be modeled in more detail with such more expressive dataflow graphs. We expect that more accurate traffic characterization than period and jitter are required to prevent very inaccurate temporal analysis results.

9. CONCLUSION

The temporal analysis flow in this paper presented a temporal analysis method which is applicable to analyze systems with non-starvation-free schedulers using dataflow analysis techniques.

The presented analysis techniques do not make assumptions on the graph topology and do therefore support cyclic graph topologies. It is also shown that buffer capacity constraints can be taken into account during the analysis and sufficient buffer sizes can be determined after the analysis. Furthermore, the correlation of event streams is preserved which is beneficial for the accuracy of the analysis method. Also a linearized variant of the analysis flow is presented. This linear variant has a polynomial time-complexity and can therefore be used to explore the design space faster.

The presented techniques are evaluated by discussing different issues complication the temporal analysis of systems with non-starvation-free schedulers. It is shown how these issues are solved and how the presented approach can improve temporal analysis results. Furthermore we have shown in the case-study how the presented analysis method can be used to improve the accuracy of the temporal analysis of a real-time stream processing system.

10. REFERENCES

[1] D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[2] S. Chakraborty, S. Künzli, and L. Thiele. A General Framework for Analysing System Properties in Platform-Based Embedded System Designs. In *Design, Automation and Test in Europe (DATE)*, pages 190–195. IEEE, 2003.

[3] ETSI. Digital audio broadcasting (DAB) to mobile, portable and fixed receivers. ETS 300 401 v1.4.1, 2006-06.

[4] A. Ghamarian et al. Throughput Analysis of Synchronous Data Flow Graphs. In *Proc of the Int'l Conf. on Application of Concurrency to System Design (ACSD)*, pages 25–36. IEEE, 2006.

[5] R. Henia et al. System Level Performance Analysis - the SymTA/S Approach. In *IEE Proceedings Computers and Digital Techniques*, volume 152, pages 148–166, 2005.

[6] B. Jonsson et al. Cyclic Dependencies in Modular Performance Analysis. In *Proc. of the ACM Int'l Conf. on Embedded Software (EMSOFT)*, pages 179–188. ACM, 2008.

[7] A.E. Kiasari, A. Jantsch, and Z. Lu. Mathematical Formalisms for Performance Evaluation of Networks-on-Chip. *ACM Computing Surveys*, Sep. 2013. (to appear).

[8] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus: a Theory of Deterministic Queuing Systems for the Internet*, volume 2050. Springer, 2001.

[9] E. Lee and T. Parks. Dataflow Process Networks. *Proceedings of the IEEE*, 83(5):773–801, 1995.

[10] A. Lele, O. Moreira, and P. Cuijpers. A New Data Flow Analysis Model for TDM. In *Proc. of the ACM Int'l Conf. on Embedded Software (EMSOFT)*, pages 237–246. ACM, 2012.

[11] O. Moreira and M. Bekooij. Self-timed scheduling analysis for real-time applications. *EURASIP Journal on Advances in Signal Processing*, 2007.

[12] O. Moreira et al. Buffer Sizing for Rate-Optimal Single-Rate Data-Flow Scheduling Revisited. *IEEE Transactions on Computers*, 59(2):188–201, 2010.

[13] K. Richter, R. Racu, and R. Ernst. Scheduling Analysis Integration for Heterogeneous Multiprocessor SoC. In *Proc. of the Int'l Real-Time Systems Symposium (RTSS)*, pages 236–245. IEEE, 2003.

[14] S. Schliecker and R. Ernst. A Recursive Approach to End-To-End Path Latency Computation in Heterogeneous Multiprocessor Systems. In *Proc. of the IEEE/ACM/IFIP Int'l Conf. on Hardware/software Codesign and System Synthesis (CODES+ISSS)*, pages 433–442. ACM, 2009.

[15] S. Sriram and S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. Marcel Dekker, Inc., 2000.

[16] L. Thiele and N. Stoimenov. Modular Performance Analysis of Cyclic Dataflow Graphs. In *Proc. of the ACM Int'l Conf. on Embedded Software (EMSOFT)*, pages 127–136. ACM, 2009.

[17] K. Tindell, A. Burns, and A. Wellings. An Extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks. *Real-Time Systems*, 6(2):133–151, 1994.

[18] M. Wiggers, M. Bekooij, and G. Smit. Efficient Computation of Buffer Capacities for Cyclo-Static Dataflow Graphs. In *Proc. of the Design Automation Conf. (DAC)*, page 663. ACM, 2007.

[19] M. Wiggers, M. Bekooij, and G. Smit. Modelling Run-Time Arbitration by Latency-Rate Servers in Dataflow Graphs. In *Proc. of the Int'l Workshop on Software & Compilers for Embedded Systems (SCOPES)*, pages 11–22. ACM, 2007.

[20] M. Wiggers, M. Bekooij, and G. Smit. Monotonicity and Run-Time Scheduling. In *Proc. of the ACM Int'l Conf. on Embedded Software (EMSOFT)*, pages 177–186. ACM, 2009.